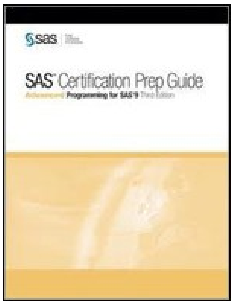


SAS Certification Prep Guide: Advanced Programming for SAS 9, Third Edition

by SAS Institute

SAS Institute. (c) 2011. Copying Prohibited.

Reprinted for Madhusmita Nayak, Accenture

madhusmita.nayak@accenture.com

Reprinted with permission as a subscription benefit of **Skillport**,
http://skillport.books24x7.com/

All rights reserved. Reproduction and/or distribution in whole or in part in electronic,paper or other forms without written permission is prohibited.

Chapter 13: Creating Samples and Indexes

Overview

Introduction

Some of the SAS data sets that you work with might be quite large. Large data sets can take a relatively long time to process because, by default, SAS reads observations in a data set sequentially. For example, assume that your data set has five hundred observations. In order to read the five-hundredth observation, SAS first reads the observations numbered 1 through 499, and then reads observation number 500. Sometimes, you might want to make SAS access specific observations directly for greater speed and efficiency.

You will need to access specific observations directly when you want to create a *representative sample* of a large data set, which can be much easier to work with than the full data set. For example, if you are concerned about the accuracy of the data in a large data set, you could audit a small sample of the data in order to determine whether a full audit is necessary. A representative sample is a *subset* of the full data set. The subset should contain observations that are taken from throughout the original data set so that the subset gives an accurate representation of the full data set. This chapter discusses two types of representative samples:

- *systematic samples*
- *random samples*.

Indexes can also make working with very large data sets easier. An index is a separate data structure that is associated with a data set, and that contains information about the specific location of observations in the data set according to the value of *key variables*. An index enables you to access a particular observation directly, without needing to read all of the observations that precede it in the data set. Indexes are useful in many instances, including WHERE and BY processing. This chapter discusses how to create and maintain both *simple* and *composite indexes*.

Objectives

In this chapter, you learn to

- create a systematic sample from a known number of observations
- create a systematic sample from an unknown number of observations
- create a random sample with replacement
- create a random sample without replacement
- use indexes
- create indexes in the DATA step
- manage indexes with PROC DATASETS
- manages indexes with PROC SQL
- document and maintain indexes.

Prerequisites

Before beginning this chapter, you should complete the following chapter:

- "Performing Queries Using PROC SQL" on page 4.

Creating a Systematic Sample from a Known Number of Observations

Overview

SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

One type of representative sample that you might want to create is a *systematic sample*. A systematic sample contains observations that are chosen from the original data set at *regular intervals*. For example, a systematic sample could contain every hundredth observation of a very large data set.

To create a systematic sample from a data set that has a known number of observations, you use the POINT= option in the SET statement.

General form, SET statement with POINT= option:

SET *data-set-name* **POINT=** *point-variable;*

where

point-variable

- names a temporary numeric variable whose value is the observation number of the observation to be read
- must be given a value before the execution of the SET statement
- must be a variable and not a constant value.

The value of the variable that is named by the POINT= option should be an integer that is greater than zero and less than or equal to the number of observations in the SAS data set. SAS uses the value to point to a specific observation in the SET statement. You must assign this value within the program so that the POINT= variable has a value when the SET statement begins execution. Also, in order for SAS to read different observations into the sample, the value of the POINT= variable must change during execution of the DATA step.

Example

You can place the SET statement with the POINT= option inside a DO loop that will assign a different value to the POINT= variable on each iteration. In the following code sample, the DO loop assigns a value to the variable **`pickit`**, which is used by the POINT= option to select every tenth observation from *Sasuser.Sale2000*. Notice that the following example is not a complete step.

```
do pickit=1 to 142 by 10;
   set sasuser.sale2000 point=pickit;
   output;
end;
```

Note In general, samples are most useful when you are working with very large data sets. However, the sample data sets that are included in this chapter are relatively small. The basics of creating samples and indexes are the same no matter the size of the data set, and you can apply the techniques in this chapter to larger data sets.

By default, SAS reads a data set sequentially, beginning with the first observation. A DATA step stops processing when SAS reaches the *end-of-file marker* after reading the last observation in the data set.

The POINT= option uses *direct-access read mode*, which means that SAS reads only those observations that you direct it to read. In direct-access read mode, SAS does not detect the end-of-file marker. Therefore, when you use the POINT= option in a SET statement, you must use a *STOP statement* to prevent the DATA step from looping continuously.

The STOP statement causes SAS to stop processing the current DATA step immediately and resume processing statements after the end of the current DATA step.

General form, STOP statement:

STOP;

Example

The *Sasuser.Revenue* data set contains 142 observations. Suppose you want to select a ten-observation subset of the data set *Sasuser.Revenue* by reading every fifteenth observation. You can use the POINT= option in a SET statement inside a DO loop to create this sample.

```
data sasuser.subset;
   do pickit=1 to 142 by 15;
      set sasuser.revenue point=pickit;
      output;
   end;
   stop;
run;

proc print data=sasuser.subset;
run;
```

The program above creates the *Sasuser.Subset* data set, shown below.

Obs	Origin	Dest	FlightID	Date	Rev1st	RevBusiness	RevEcon
1	ANC	RDU	IA03400	02DEC1999	15829	28420	68688
2	CCU	PEK	IA09700	08DEC1999	19992	27832	36010
3	DEL	JRS	IA09001	27DEC1999	14434	16169	42066
4	FRA	RDU	IA00400	18DEC1999	22893	28824	87750
5	HKG	SYD	IA10100	24DEC1999	34074	39990	101898
6	HNL	SFO	IA03000	30DEC1999	10868	16825	46248
7	LHR	JED	IA08100	21DEC1999	21942	42330	63990
8	PEK	CCU	IA09801	11DEC1999	23324	27264	37949
9	RDU	LHR	IA00101	17DEC1999	17600	30520	79119
10	SIN	CCU	IA09401	23DEC1999	20061	24843	36556

Creating a Systematic Sample from an Unknown Number of Observations

Overview

Sometimes you might not know how many observations are in the original data set from which you want to create a systematic sample. In order to make a systematic sample, you need to know the total number of observations in the original data set so that you can choose observations that are evenly distributed from it.

You can use the NOBS= option in the SET statement to determine how many observations there are in a SAS data set.

General form, SET statement with NOBS= option:

SET *SAS-data-set* **NOBS**=*variable;*

where

variable

names a temporary numeric variable whose value is the number of observations in the input data set.

Note If *multiple data sets* are listed in the SET statement, the value of the NOBS=variable is the *total number* of observations in all of the data sets that are listed.

The value of the NOBS= variable is assigned automatically during compilation when SAS reads the descriptor portion of the data file. Therefore, this value is a vailable at any time during execution of the DATA step.

Note The total that is used as a value for the NOBS= variable includes observations that have been marked for deletion but have not been physically removed from the data set.

You can use the NOBS= option in conjunction with the POINT= option to create a systematic sample of a data set if you do not know how many observations are in the data set.

Example

Suppose you want to create a systematic sample of the *Sasuser.Revenue* data set, and you do not know how many observations are in it. In the following example, **totobs** is assigned the value of the total number of observations in the data set *Sasuser.Revenue* during compilation. Then, **totobs** is used as the upper limit for the DO loop that controls how many observations are chosen for the systematic sample.

```
data sasuser.subset;
   do pickit=1 to totobs by 10;
      set sasuser.revenue point=pickit nobs=totobs;
      output;
   end;
   stop;
run;
```

The resulting *Sasuser.Subset* data set contains every tenth observation from the *Sasuser.Revenue* data set. When the program above is submitted, the DATA step iterates only once, and the DO loop iterates multiple times within the DATA step.

Creating a Random Sample with Replacement

Another type of representative sample that you might want to create is a *random sample*. A random sample contains observations that are chosen from the original data set on a random basis. When you create a random sample *with replacement*, it is possible for one observation to appear in the sample multiple times. You can think of the original data set as a pool of possible observations that might be chosen for inclusion in the sample. For each observation in the sample data set, SAS chooses an observation randomly from the original pool, copies it to the sample data set, and replaces it in the pool.

Using the RANUNI Function

In order to create a random sample, you need to generate a random number. SAS provides several random number functions to generate random numbers from various distributions. One example of a random number function is the RANUNI function.

General form, RANUNI function:

RANUNI *(seed)*

where

seed

is a nonnegative integer with a value less than 2^{31}-1 (2,147,483,647).

The RANUNI function generates streams of random numbers from an initial starting point, called the *seed*. If you use a positive seed, you can always replicate the stream of random numbers by using the same DATA step. If you use *0* as the seed, the computer clock initializes the stream, and the stream of random numbers is not replicable.

The numbers that the RANUNI function returns are all between 0 and 1 (noninclusive). Examples of the type of number RANUNI returns include .01253689 and .95196500. If you use RANUNI in a DATA step that generates only one random number, RANUNI returns only the first number from the stream. If you use RANUNI in a DATA step that generates multiple numbers, such as in a DO loop, RANUNI will return a different random number each time the loop iterates.

Here are some examples.

Example

The following DATA step creates one observation with one variable named **`varone`** and assigns a random number to it. You can submit this DATA step multiple times, or in multiple SAS sessions, and **`varone`** will have the same value.

```
data random1;
   varone=ranuni(10);
run;
```

The following DATA step creates ten observations with one variable named **`varone`** and assigns a random number as a value for **`varone`**. You can submit this DATA step multiple times, or in multiple SAS sessions, and **`varone`** will have the same ten values.

```
data random2;
   do i=1 to 10 by 1;
      varone=ranuni(10);
      output;
   end;
run;
```

If you changed the seed value from *10* toa different value in either of the two DATA steps above, the values for **`varone`** would be different when you submitted the DATA step than it was when the seed value was *10*. However, **`varone`** will have the same ten values each time you submit the DATA step with a constant seed value.

> **Note** For clarity and consistency in the sample programs, all remaining examples of the RANUNI function in this chapter will use a seed of *0*.

Using a Multiplier with the RANUNI Function

By default, RANUNI generates numbers that are between 0 and 1. To increase the interval from which the random number is chosen, you use a *multiplier* on the RANUNI function. For example, if you want to generate a random number between 0 and 50, you use the following code:

```
ranuni(0)*5 0
```

You have seen that the RANUNI function generates a random number between 0 and 1. However, in order to create a random sample, you need to generate a random integer that will match one of the observation numbers in the original data set.

Using the CEIL Function

You can use the CEIL function in conjunction with the RANUNI function to generate a random integer.

General form, CEIL function:

CEIL *(argument)*

where

argument

is numeric.

The CEIL function returns the smallest integer that is greater than or equal to the argument. Therefore, if you apply the CEIL function to the result of the RANUNI function, you can generate a random integer.

Example

The following example creates a random integer between 1 and 50:

```
ceil(ranuni(0)*50)
```

Now that you have seen how to use the CEIL function in conjunction with the RANUNI function, let us consider how to use these functions in a DATA step to create a random sample. You use the CEIL and RANUNI functions together to generate a random integer that is assigned as a value for the variable to which the POINT= option points.

SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

Example

In the following example, the CEIL and RANUNI functions are used together in the assignment statement for **`pickit`**, which is the variable that is pointed to by the POINT= option. The NOBS= option assigns the total number of observations in the *Sasuser.Revenue* data set as a value for the **`totobs`** variable. The variable **`totobs`** is then used as the multiplier in the CEIL function, so that for each iteration of the DO loop, every observation in *Sasuser.Revenue* has an equal chance of being picked for inclusion in the sample.

```
data work.rsubset (drop=i sampsize);
   sampsize=10;
   do i=1 to sampsize;
      pickit=ceil(ranuni(0)*totobs);
      set sasuser.revenue point=pickit nobs=totobs;
      output;
   end;
   stop;
run;

proc print data=work.rsubset label;
   title 'A Random Sample with Replacement';
run;
```

Since the program uses a seed of *0* for the RANUNI function, the *Work.Rsubset* data set will be different each time you submit this code. Here is an example of the possible output.

A Random Sample with Replacement

Obs	Origin	Dest	FlightID	Date	Rev1st	RevBusiness	RevEcon
1	SYD	HKG	IA10201	13DEC1999	32181	39990	111333
2	HKG	HND	IA10901	05DEC1999	17760	20160	32718
3	HKG	HND	IA10900	10DEC1999	19980	21672	36900
4	CCU	HKG	IA09901	13DEC1999	16272	20790	32400
5	RDU	ANC	IA03300	30DEC1999	17268	29400	58671
6	WLG	CBR	IA10600	08DEC1999	15496	17908	29106
7	RDU	LHR	IA00100	22DEC1999	22400	31610	77526
8	RDU	LHR	IA00100	10DEC1999	20800	32700	67968
9	FRA	RDU	IA00400	30DEC1999	24654	30025	75465
10	HND	HKG	IA11000	02DEC1999	19240	22680	34932

Creating a Random Sample without Replacement

You can also create a random sample *without replacement*. A sample without replacement *cannot* contain duplicate observations because after an observation is chosen from the original data set and output to the sample, it is programmatically excluded from being chosen again.

Example

You can use a DO WHILE loop to avoid replacement as you create your random sample. In the following example

- *Sasuser.Revenue* is the original data set.
- **`sampsize`** is the number of observations to read into the sample.
- *Work.Rsubset* is the data set that contains the random sample that you are creating.
- **`obsleft`** is the number of observations in the original data set that have not yet been considered for selection.
- **`totobs`** is the total number of observations in the original data set.
- **`pickit`** is the number of the observation to be read into the sample data set (if the RANUNI expression is true), and its

starting value is *0*.

With each iteration of the DO loop, **pickit** is incremented by *1*. If the RANUNI expression is true, the observation that is indicated by the value of **pickit** is selected for the sample, and **sampsize** is decreased by *1*. If the RANUNI expression is not true, the observation that is indicated by the value of **pickit** is not added to the sample. On each iteration of the loop, **obsleft** is decreased by *1* regardless of whether the observation is selected for the sample. The process ends when the value of **sampsize** is *0* and no additional observations are needed.

```
data work.rsubset(drop=obsleft sampsize);
   sampsize=10;
   obsleft=totobs;
   do while(sampsize>0);
      pickit+1;
      if ranuni(0)<sampsize/obsleft then do;
         set sasuser.revenue point=pickit
             nobs=totobs;
         output;
         sampsize=sampsize-1;
      end;
      obsleft=obsleft-1;
   end;
   stop;
run;

proc print data=work.rsubset label;
   title 'A Random Sample without Replacement';
run;
```

Note that each observation is considered for selection once and only once.

Because the program above uses a seed of *0* for the RANUNI function, *Work.Rsubset* will contain different observations each time you run this code. Here is an example of the possible output.

A Random Sample without Replacement

Obs	Origin	Dest	FlightiD	Date	Rev1st	RevBusiness	RevEcon
1	CCU	HKG	IA09901	13DEC1999	16272	20790	32400
2	CCU	SIN	IA09301	05DEC1999	20061	21801	33592
3	DXB	FRA	IA07800	22DEC1999	18630	40608	53148
4	DXB	FRA	IA07801	29DEC1999	18630	38070	63448
5	FRA	RDU	IA00401	25DEC1999	21132	28824	85410
6	HND	SFO	IA11101	23DEC1999	40337	46304	133245
7	LHR	JED	IA08100	09DEC1999	20723	40670	54270
8	LHR	JNB	IA08301	09DEC1999	41940	76224	111456
9	SFO	HNL	IA02900	02DEC1999	13832	20190	42640
10	SIN	CCU	IA09401	23DEC1999	20061	24843	36556

Using Indexes

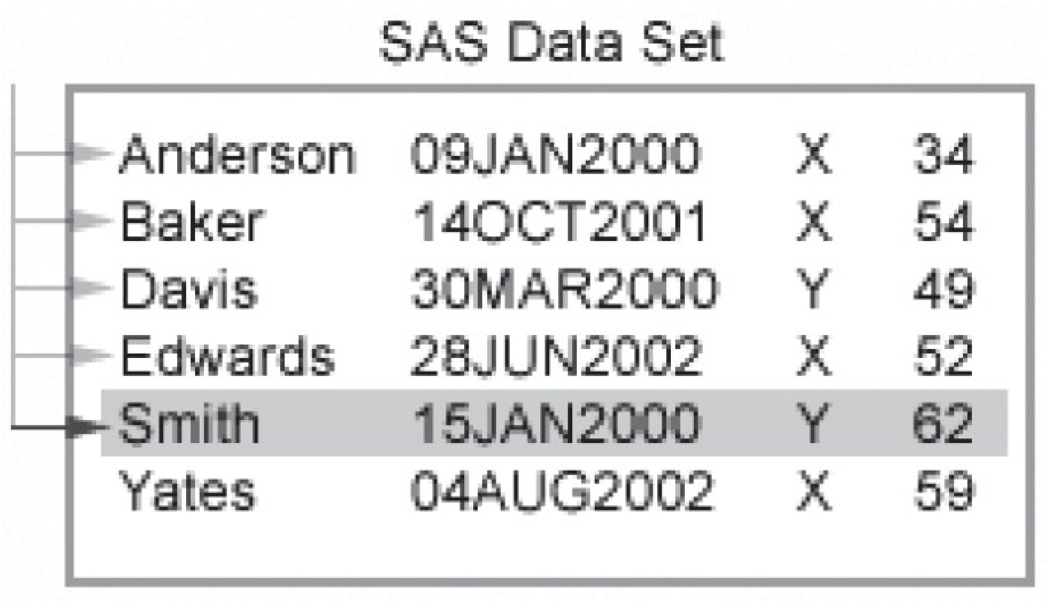
Overview

An *index* can help you quickly locate one or more particular observations that you want to read. An index is an optional file that you can create for a SAS data set in order to specify the location of observations based on values of one or more *key variables*. Indexes can provide direct access to observations in SAS data sets to

- yield faster access to small subsets of observations for WHERE processing
- return observations in sorted order for BY processing
- perform table lookup operations

- join observations
- modify observations.

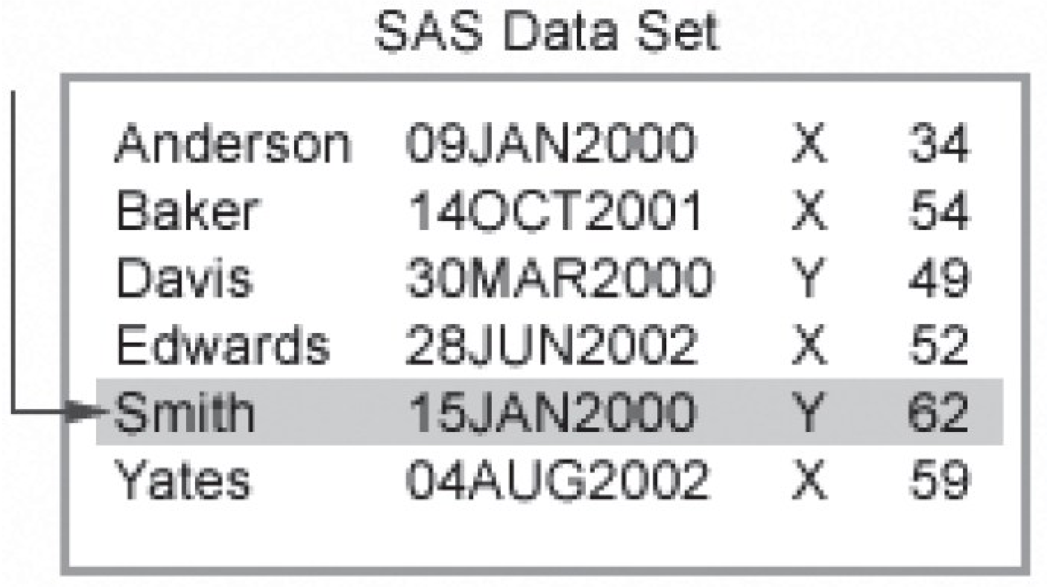
Without an index, SAS accesses observations sequentially, in the order in which they are stored in a data set. For example, if you want to access the observation in the sample SAS data set shown below that has a value of *Smith* for the variable **`Name`**, SAS begins with the first observation and reads through each one until it reaches the observation that satisfies the condition.



SAS Data Set

Anderson	09JAN2000	X	34
Baker	14OCT2001	X	54
Davis	30MAR2000	Y	49
Edwards	28JUN2002	X	52
Smith	15JAN2000	Y	62
Yates	04AUG2002	X	59

An index stores values in ascending value order for a specific variable or variables and includes information about the location of those values within observations in the data file. That is, an index consists of *value/identifierpairs* that enable you to locate an observation by value. For example, if you create an index on the sample SAS data set shown below based on the variable **`Name`**, SAS uses the index to find the observation that has a value of *Smith* for **`Name`** directly without having to read all the prior observations.



SAS Data Set

Anderson	09JAN2000	X	34
Baker	14OCT2001	X	54
Davis	30MAR2000	Y	49
Edwards	28JUN2002	X	52
Smith	15JAN2000	Y	62
Yates	04AUG2002	X	59

Types of Indexes

You can create two types of indexes:

SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

- a simple index
- a composite index.

A *simple index* consists of the values of one key variable, which can be character or numeric. When you create a simple index, SAS assigns the name of the key variableas the name of the index.

A *composite index* consists of the values of multiple key variables, which can be character, numeric, or a combination. The values of these key variables are concatenated to form a single value. For example, if an index is built on the key variables **`Lastname`** and **`Firstname`**, a value for the index is composed of the value for **`Lastname`** followed by the value for **`Firstname`**. When you create a composite index, you must specify a *unique* index name that is not the name of any existing variable or index in the data set.

Often, only the first variable of a composite index is used. For example, you could use the composite index specified in the example above **(`Lastname`** plus **`Firstname`)** for a WHERE expression that uses only **`Lastname`**. For example, the expression **`where Lastname='Smith'`** uses the composite index because **`Lastname`** is the first variable in the index. That is, the value for **`Lastname`** is the first part of the value listed in the index.

Creating Indexes in the DATA Step

Overview

To create an index at the same time that you create a data set, use the *INDEX= data set option* in the DATA statement.

General form, DATA statement with the INDEX= option:

```
DATA SAS-data-file-name (INDEX=
        (index-specification-l</UNlQUE><...index-specification-n>
        </UNIQUE>));
```

where

SAS-data-file-name

is a valid SAS data set name.

index-specification

for a simple index is the name of the key variable.

index-specification

for a composite index is *(index-name=(variable-l…variable-n))*.

UNIQUE

option specifies that values for the key variable must be unique for each observation.

Note SAS stores the name of a composite index exactly as you specify it in the INDEX= option. Therefore, if you want the name of your index to begin with a capital letter, you must specify the name with an initial capital letter in the INDEX= option.

You can create *multiple indexes* on a single SAS data set. However, keep in mind that creating and storing indexes does use system resources. Therefore, you should create indexes only on variables that are commonly used in a WHERE condition or on variables that are used to combine SAS data sets.

Note You can create an index on a SAS data file but not on a SAS data view.

The UNIQUE option guarantees that values for the key variable or the combination of a composite group of variables remain unique for every observation in the data set. In an existing data set, if the variable(s) on which you attempt to create a unique index has duplicate values, the index is not created. Similarly, if an update tries to add a record with a duplicate

SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

value for the index variable to that data set, the update is rejected. You will see examples of up dating and maintaining indexes later in this chapter.

Examples

The following example creates a simple index on the *Simple* data set. The index is named *Division*, and it contains values of the **Division** variable.

```
data simple (index=(division));
   set sasuser.empdata;
run;
```

The following example creates two simple indexes on the *Simple2* data set. The first index is named *Division*, and it contains values of the **Division** variable. The second index is called *EmpID*, and it contains unique values of the **EmpID** variable.

```
data simple2 (index=(division empid/unique));
   set sasuser.empdata;
run;
```

The following example creates a composite index on the *Composite* data set. The index is named *Empdiv*, and it contains concatenated values of the **Division** variable and the **EmpID** variable.

```
data composite (index=(Empdiv=(division empid)));
   set sasuser.empdata;
run;
```

When you create or use an index, you might want to verify that it has been created or used correctly. To display information in the SAS log concerning index creation or index usage, set the value of the *MSGLEVEL=* system option to I.

General form, MSGLEVEL= system option:

OPTIONS MSGLEVEL= N|I;

where

N

prints notes, warnings, and error messages only. This is the default.

I

prints additional notes or INFO messages pertaining to index usage, merge processing, and sort utilities along with standard notes, warnings, and error messages.

Example

The following code sets the MSGLEVEL= system option to I and creates the *Sasuser.Sale2000* data set with two indexes:

```
options msglevel=i;
data sasuser.sale2000(index=(origin
     flightdate=(flightid date)/unique));
   infile sale2000 dsd;
   input FlightID $ RouteID $ Origin $
         Dest $ Cap1st CapBusiness
         CapEcon CapTotal CapCargo
         Date Psgr1st PsgrBusiness
         PsgrEcon Rev1st RevBusiness
         RevEcon SaleMon $ CargoWgt
         RevCargo;
   format date date9.;
run;
```

Here are the messages that are written to the SAS log when the program above is submitted.

Table 13.1: SAS Log

```
NOTE: The infile SALE2000 is:
      File Name=C:\My SAS Files\9.0\sale2000.dat,
      RECFM=V,LRECL=256

NOTE: 153 records were read from the infile SALE2000.
      The minimum record length was 82.
      The maximum record length was 100.
NOTE: The data set SASUSER.SALE2000 has 153 observations
      and 19 variables.
NOTE: Composite index flightdate has been defined.
NOTE: Simple index origin has been defined.
NOTE: DATA statement used (Total process time):
      real time           1.08 seconds
      cpu time            0.04 seconds
```

Determining Whether SAS Is Using an Index

It is not always possible or more efficient for SAS to use an existing index to access specific observations directly. An index is *not used*

- with a subsetting IF statement in a DATA step
- with particular WHERE expressions
- if SAS determines it is more efficient to read the data sequentially.

Example

You can use the MSGLEVEL= option to determine whether SAS is using an index. The following SAS log messages show examples of the INFO messages that indicate whether an index was used.

Table 13.2: SAS Log

```
6 options msglevel=i;
7
8 proc print data=sasuser.revenue;
9    where flightid ne 'IA11200';
INFO: Index FlightID not used. Increasing bufno to 3 may help.
```

Table 13.3: SAS Log

```
11 options msglevel=i;
12
13 data somflights;
14    set sasuser.revenue;
15    where flightid > 'IA11200';
INFO: Index FlightID selected for WHERE clause optimization.
```

Managing Indexes with PROC DATASETS

Overview

You have seen how to create an index at the same time that you create a data set. You can also create an index on an existing data set, or delete an index from a data set. One way to accomplish either of these tasks is to rebuild the data set. However, rebuilding the data set is not the most efficient method for managing indexes.

You can use the DATASETS procedure to manage indexes on an existing data set. This uses fewer resources than it would to rebuild the data set. You use the *MODIFY statement* with the *INDEXCREATE statement* to create indexes on a data set. You use the MODIFY statement with the *INDEXDELETE statement* to delete indexes from a data set. You can also use the INDEX CREATE statement and the INDEX DELETE statement in the same step.

General form, PROC DATASETS to create and delete an index:

```
PROC DATASETS LIBRARY= libref<NOLIST>;
        MODIFY SAS-data-set-name;
        INDEX DELETE index-name;
        INDEX CREATE index-specification;
QUIT;
```

where

libref

points to the SAS library that contains *SAS-data-set-name*.

NOLIST

option suppresses the printing of the directory of SAS files in the SAS log and as ODS output.

index-name

is the name of an existing index to be deleted.

index-specification

for a simple index is the name of the key variable.

index-specification

for a composite index is *index-name=(variable-l…variable-n)*.

The INDEX CREATE statement in PROC DATASETS *cannot* be used if the index to be created already exists. In this case, you must delete the existing index of the same name, and then create the new index.

Tip PROC DATASETS executes statements in order. Therefore, if you are deleting and creating indexes in the same step, you should delete indexes first so that the newly created indexes can reuse space of the deleted indexes.

Example

The following example creates an index named *Origin* on the *Sasuser.Sale2000* data set. *Origin* is a simple index that is based on the key variable **`Origin`**.

```
proc datasets library=sasuser nolist;
   modify sale2000;
   index create origin;
quit;
```

The following example first deletes the *Origin* index from the *Sasuser.Sale2000* data set, and creates two new indexes on the *Sasuser.Sale2000* data set. *FlightID* is a simple index that is based on the values of the key variable **`FlightID`**. *Fromto* is a composite index that is based on the concatenated values of the key variables **`Origin`** and **`Dest`**.

```
proc datasets library=sasuser nolist;
   modify sale2000;
   index delete origin;
   index create flightid;
   index create Fromto=(origin dest);
quit;
```

Managing Indexes with PROC SQL

Overview

You can also create indexes on or delete indexes from an existing data set within a PROC SQL step. The *CREATE INDEX statement* enables you to create an index on a data set. The *DROP INDEXstatement* enables you to delete an index from a data set.

SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

General form, PROC SQL to create and delete an index:

```
PROC SQL;
     CREATE <UNIQUE> INDEX index-name
          ON table-name(column-name-1<…,column-name-n>);
     DROP INDEX index-name FROM table-name;
QUIT;
```

where

index-name

is the same as *column-name-1* if the index is based on the values of one column only.

index-name

is not the same as any *column-name* if the index is based on multiple columns.

table-name

is the name of the data set to which *index-name* is associated.

Example

The following example creates a simple index named *Origin* on the *Sasuser.Sale2000* data set. The index is based on the values of the **Origin** column.

```
proc sql;
   create index origin on sasuser.sale2000(origin);
quit;
```

The following example deletes the *Origin* index from the *Sasuser.Sale2000* data set and creates a new index named *Tofrom* that is based on the concatenation of the values from the columns **Origin** and **Dest:**

```
proc sql;
   create index Tofrom
      on sasuser.sale2000(origin, dest);
   drop index origin from sasuser.sale2000;
quit;
```

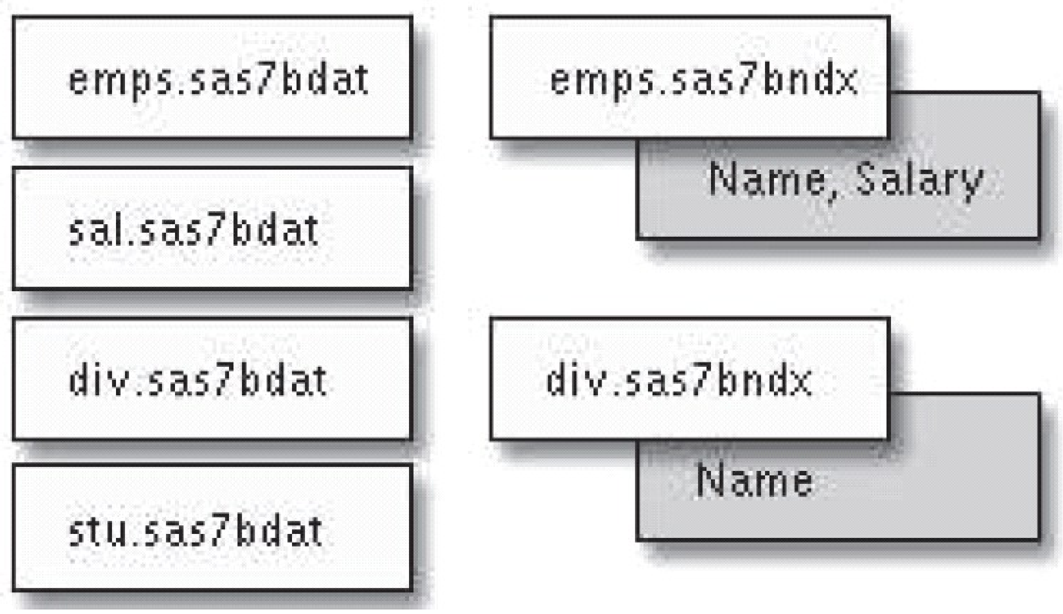
Documenting and Maintaining Indexes

Overview

Indexes are stored in the same SAS library as the data set that they index, but in a separate SAS file from the data set. Index files have the *same name* as the associated data file, and have a member type of INDEX. There is only one index file per data set; all indexes for a data set are stored together in a single file.

The following image shows the relationship of SAS data set files and SAS index files in a Windows operating environment. Notice that although they have different file extensions, the index files have the same name as the data set with which they are associated. Also, notice that each index file can contain one or more indexes, and that different index files can contain indexes with identical names.

SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited



Note Although index files are stored in the same location as the data sets to which they are associated

- index files do not appear in the SAS Explorer window
- index files do not appear as separate files in z/OS operating environment file lists.

Sometimes, you might want to view a list of the indexes that exist for a data set. You might also want to see information about the indexes such as whether they are unique, and what key variables they use. Let us consider some ways to document indexes.

Information about indexes is stored in the descriptor portion of the data set. You can use either the *CONTENTS procedure* or the *CONTENTS statement in PROC DATASETS* to list information from the descriptor portion of a data set.

Output from the CONTENTS procedure or from the CONTENTS statement in PROC DATASETS contains the following information about the data set:

- general and summary information
- engine/host dependent information
- alphabetic list of variables and attributes
- alphabetic list of integrity constraints
- *alphabetic list of indexes and attributes*.

General form, PROC CONTENTS:

PROC CONTENTS DATA=*<libref.>SAS-data-set-name;*

RUN;

where

SAS-data-set-name

specifies the data set for which the information will be listed.

General form, PROC DATASETS with the CONTENTS statement:

```
PROC DATASETS <LIBRARY=libref> <NOLIST>;
     CONTENTS DATA=<libref.>SAS-data-set-name;
QUIT;
```

where

SAS-data-set-name

specifies the data set for which the information will be listed.

NOLIST

option suppresses the printing of the directory of SAS files in the SAS log and as ODS output.

Note If you use the *LIBRARY*=option, you do not need to specify a libref in the *DATA*= option. Likewise, if you specify a *libref in* the *DATA*= option, you do not need to use the *LIBRARY*=option.

Example

The following example prints information about the *Sasuser.Sale2000* data set. Notice that the library is specified in the LIBRARY=option of the PROC DATASETS statement.

```
proc datasets library=sasuser nolist;
   contents data=sale2000;
quit;
```

The following example also prints information about the *Sasuser.Sale2000* data set. Notice that the library is specified in the CONTENTS statement.

```
proc datasets nolist;
   contents data=sasuser.sale2000;
quit;
```

The following example also prints information about the *Sasuser.Sale2000* data set:

```
proc contents data=sasuser.sale2000;
run;
```

The PROC DATASETS and PROC CONTENTS output from these programs is identical. The last piece of information printed in each set of output is a list of the indexes that have been created for Sasuser .>Sa/e2000, as shown below.

Alphabetic List of Indexes and Attributes

#	Index	Unique Option	# of Unique Values	Variables
1	Origin		32	
2	flightdate	YES	156	FlightID Date

You can also use either of these methods to list information about an *entire SAS library* rather than an individual data set. To list the contents of all files in a SAS library with either PROC CONTENTS or with the CONTENTS statement in PROC DATASETS, you specify the *keyword* _ALL_ in the DATA= option.

Example

The following example prints information about all of the files in the *Work* data library:

```
proc contents data=work._all_;
run;
```

The following example also prints information about all of the files in the *Work* data library:

```
proc datasets library=work nolist;
   contents data=_all_;
quit;
```

Remember that indexes are stored in a separate SAS file. When you perform maintenance tasks on a data set, there might be resulting effects on the index file. If you alter the variables or values within a data set, there might be a resulting effect on the value/identifier pairs within a particular index.

The following table describes the effects on an index or an index file that result from several common maintenance tasks.

Task	Effect
Add observation(s) to data set	Value/identifier pairs are added to index(es).
Delete observation(s) from data set	Value/identifier pairs are deleted from index(es).
Update observation(s) in data set	Value/identifier pairs are updated in index(es).
Delete data set	The index file is deleted.
Rebuild data set with DATA step	The index file is deleted.
Sort the data in place with the FORCE option in PROC SORT	The index file is deleted.

Let us consider some of the other common tasks that you might perform on your data sets, as well as the actions that SAS performs on the index files as a result.

Copying Data Sets

You might want to copy an indexed data set to a new location. You can copy a data set with the *COPYstatement* in a PROC DATASETS step. When you use the COPY statement to copy a data set that has an index associated with it, a new index file is automatically created for the new data file.

General form, PROC DATASETS with the COPY statement:

```
PROC DATASETS LIBRARY=old-libref<NOLIST>;
     COPY OUT=new-libref;
     SELECT SAS-data-set-name;
QUIT;
```

where

old-libref

names the library from which the data set will be copied.

new-libref

names the library to which the data set will be copied.

SAS-data-set-name

names the data set that will be copied.

You can also use the *COPYprocedure* to copy data sets to a new location. Generally, PROC COPY functions the same as the COPY statement in the DATASETS procedure. When you use PROC COPY to copy a data set that has an index associated with it, a new index file is automatically created for the new data file. If you use the MOVE option in the COPY procedure, the index file is deleted from the original location and rebuilt in the new location.

General form, PROC COPY step:

```
PROC COPY OVT=new-libref IN=old-libref
     <MOVE>;
     SELECT SAS-data-set-name(s);
RUN;
QUIT;
```

where

old-libref

names the library from which the data set will be copied.

new-libref

names the library to which the data set will be copied.

SAS-data-set-name

names the data set or data sets that will be copied.

Examples

The following programs produce the same result. Both programs copy the *Sale2000* data set from the *Sasuser* library and place it in the *Work* library. Likewise, both of these programs cause a new index file to be created for *Work.Sale2000* that contains all indexes that exist in *Sasuser.Sale2000*.

```
proc datasets library=sasuser nolist;
   copy out=work;
   select sale2000;
quit;

proc copy out=work in=sasuser;
   select sale2000;
run;
```

Note If you copy and paste a data set in either *SASExplorer* or in *SASEnterprise Guide*, a new index file is automatically created for the new data file.

Renaming Data Sets

Another common task is to rename an indexed data set. To preserve the index, you can use the *CHANGE statement* in PROC DATASETS to rename a data set. The index file will be automatically renamed as well.

General form, PROC DATESETS with the CHANGE statement:

```
PROC DATASETS LIBRARY=libref <NOLIST>;
     CHANGE old-data-set-name =new-data-set-name;
QUIT;
```

where

libref

names the SAS library where the data set is stored.

old-data-set-name

is the current name of the data set.

new-data-set-name

is the new name of the data set.

Example

The following example copies the *Revenue* data set from *Sasuser* into *Work*, and renames the *Work.Revenue* data set to *Work.Income*. The index file that is associated with *Work.Revenue* is also renamed to *Work.Income*.

```
proc copy out=work in=sasuser;
   select revenue;
run;

proc datasets library=work nolist;
```

SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

```
   change revenue=income;
quit;
```

Renaming Variables

You have seen how to use PROC DATASETS to rename an indexed data set. Similarly, you might want to rename one or more variables within an indexed data set. In order to preserve any indexes that are associated with the data set, you can use the *RENAME statement* in the DATASETS procedure to rename variables.

General form, PROC DATASETS with the RENAME statement:

```
PROC DATASETS LIBRARY=libref <NOLIST>;
        MODIFY=SAS-data-set-name;
        RENAME Sold-var-name-1 = new-var-name-1
        <…old-var-name-n = new-var-name-n>;
QUIT;
```

where

libref

names the SAS library where the data set is stored.

SAS-data-set-name

is the name of the data set that contains the variables to be renamed.

old-var-name

is the original variable name.

new-var-name

is the new name to be assigned to the variable.

When you use the RENAME statement to change the name of a variable for which there is a simple index, the statement also renames the index. If the variable that you are renaming is used in a composite index, the composite index automatically references the new variable name. However, if you attempt to rename a variable to a name that has already been used for a composite index, you will receive an error message.

Example

The following example renames the variable **`FlightID`** as **`FlightNum`** in the *Work.Income* data set. If a simple index exists that is named *FlightID*, the index will be renamed *FlightNum*.

```
proc datasets library=work nolist;
   modify income;
   rename flightid=FlightNum;
quit;
```

Summary

Contents

This section contains the following topics.

Text Summary

Creating a Systematic Sample from a Known Number of Observations

Sometimes you might want to create a representative sample of a large data set. One type of representative sample, called a systematic sample, contains observations that are chosen from the original data set at regular intervals. You can use the POINT= option in the SET statement to make SAS read a specific observation into the sample. Since SAS uses direct-access read mode with the POINT= option, you must use a STOP statement to prevent the DATA step from looping continuously.

Creating a Systematic Sample from an Unknown Number of Observations

You might want to create a systematic sample from a data set that contains an unknown number of observations. In order to be sure that your sample observations are chosen from regular intervals across the entire original data set, you need to know how many observations are in the data set. You can use the NOBS= option in the SET statement to determine how many observations are in the input data set. You can use the NOBS= option in conjunction with the POINT= option to direct SAS to read specific observations that will form a systematic sample.

Creating a Random Sample with Replacement

Another type of representative sample that you might want to create is a random sample, in which observations are chosen randomly from the original data set. You can use the RANUNI function in conjunction with the CEIL function to generate a random integer. With a random integer, you can direct SAS to read a specific (but random) observation into the sample. When you create a random sample with replacement, each observation in the original data set has an equal chance of being chosen for inclusion in the sample each time SAS chooses an observation. That is, in a random sample with replacement, one observation might be chosen from the original data set and included in the sample multiple times.

Creating a Random Sample without Replacement

You can also create a random sample without replacement. This means that once an observation has been included in the sample it is no longer eligible to be chosen again. You can use a DO WHILE loop to prevent replacement in your random samples.

Using Indexes

An index is a SAS file that is associated with a data set and that contains information about the location and the values of key variables in the data set. Indexes enable SAS to directly access specific observations rather than having to read all observations sequentially. An index can be simple or composite.

Creating Indexes in the DATA Step

You can create an index at the same time that you create a data set by using the INDEX= option in the DATA statement. Both simple and composite indexes can be unique, if there are no duplicate values for any key variable in the data set. You can create multiple indexes on one data set. You can use the MSGLEVEL= system option to write informational messages to the SAS log that pertain to indexes. Indexes can improve the efficiency of SAS, but there are certain instances where indexes will not improve efficiency and therefore will not be used.

Managing Indexes with PROC DATASETS and PROC SQL

You can use the INDEX CREATE statement or the INDEX DELETE statement in PROC DATASETS to create an index on or delete an index from an existing data set. Using PROC DATASETS to manage indexes uses less system resources than it would to rebuild the data set and update indexes in the DATA step. If you want to delete an index and create an index in the same PROC DATASETS step, you should delete the old index before you create the new index so that SAS can reuse space from the deleted index. You can also use PROC SQL to create an index on or delete an index from an existing data set.

Documenting and Maintaining Indexes

All indexes that are created for a particular data set are stored in one file in the same SAS library as the data set. You can use PROC CONTENTS to print a list of all indexes that exist for a data set, along with other information about the data set. The CONTENTS statement of the PROC DATASETS step can generate the same list of indexes and other information about a data set.

Many of the maintenance tasks that you perform on your data sets will affect the index file that is associated with the data

SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

set. When you copy a data set with the COPY statement in PROC DATASETS, the index file is reconstructed for you. When you rename a data set or rename a variable with PROC DATASETS, the index file is automatically updated to reflect this change.

Syntax

```
DATA SAS-data-set-name;
     point-variable=CEIL(RANVNI(seed) *nobs-variable);
     SET SAS-data-set-name POINT=point-variable NOBS=nobs-variable;
     STOP;
RUN;
OPTIONS MSGLEVEL= N|I;
DATA SAS-data-file-name (INDEX=
     (index-specification-1 <UNIQUE><…index-specification-n</UNlQOE>>));
     SET SAS-data-set-name;
RUN;
PROC DATASETS LIBRARY=libref<NOLIST>;
     MODIFY SAS-data-set-name;
     INDEX DELETE index-name;
     INDEX CREATE index-specification;
QUIT;
PROC SQL;
     CREATE <UNIQUE> INDEX index-name
         ON table-name{column-name-l<…,column-name-n>);
     DROP INDEX index-name FROM table-name;
QUIT;
PROC CONTENTS DATA=<libref.>SAS-data-set-name;
RUN;
PROC DATASETS <LIBRARY=libref> <NOLIST>;
     CONTENTS DATA=<libref.>SAS-data-set-name;
QUIT;
PROC DATASETS LIBRARY=old-libref<NOLIST>;
     COPY OUT=new-libref;
     SELECT SAS-data-set-name;
QUIT;
PROC COPY OUT=new-librefIN=old-libref<MOVE>;
     SELECT SAS-data-set-name(s);
RUN;
QUIT;
PROC DATASETS LIBRARY=libref<NOLIST>;
     CHANGE old-data-set-name - new-data-set-name;
QUIT;
PROC DATASETS LIBRARY=libref<NOLIST>;
     MODIFY SAS-data-set-name;
     RENAME old-var-name-1 - new-var-name-1
    <…old-var-name-n - new-var-name-n>;
QUIT;
```

Sample Programs

Creating a Systematic Sample from a Known Number of Observations

```
data sasuser.subset;
   do pickit=1 to 142 by 15;
      set sasuser.revenue point=pickit;
      output;
   end;
   stop;
run;
```

Creating a Systematic Sample from an Unknown Number of Observations

```
data sasuser.subset;
   do pickit=1 to totobs by 10;
      set sasuser.revenue point=pickit
          nobs=totobs;
      output;
   end;
   stop;
run;
```

Creating a Random Sample with Replacement

```
data work.rsubset (drop=i sampsize);
   sampsize=10;
   do i=1 to sampsize;
      pickit=ceil(ranuni(0)*totobs);
      set sasuser.revenue point=pickit
          nobs=totobs;
      output;
   end;
   stop;
run;

proc print data=work.rsubset label;
   title 'A Random Sample with Replacement';
run;
```

Creating a Random Sample without Replacement

```
data work.rsubset(drop=obsleft sampsize);
   sampsize=10;
   obsleft=totobs;
   do while(sampsize>0);
      pickit+1;
      if ranuni(0)<sampsize/obsleft then do;
         set sasuser.revenue point=pickit
             nobs=totobs;
         output;
         sampsize=sampsize-1;
      end;
   obsleft=obsleft-1;
   end;
   stop;
run;

proc print data=work.rsubset heading=h label;
title 'A Random Sample without Replacement';
run;
```

Creating an Index in the DATA Step

```
options msglevel=i;
data sasuser.sale2000(index=(origin FlightDate=
                             (flightid date)/unique));
infile 'sale2000.dat';
input FlightID $7. RouteID $7. Origin $3.
      Dest $3. Cap1st 8. CapBusiness 8.
      CapEcon 8. CapTotal 8. CapCargo 8.
      Date date9. Psgr1st 8./
      PsgrBusiness 8. PsgrEcon 8.
      Rev1st dollar15.2
      RevBusiness dollar15.2
      RevEcon dollar15.2 SaleMon $7.
      CargoWgt 8./ RevCargo dollar15.2;
run;
```

Managing Indexes with PROC DATASETS

```
proc datasets library=sasuser nolist;
   modify sale2000;
   index delete origin;
   index create flightid;
   index create Tofrom=(origin dest);
quit;
```

Managing Indexes with PROC SQL

```
proc sql;
   create index Tofrom on
          sasuser.sale2000(origin, dest);
   drop index origin from sasuser.sale2000;
quit;
```

You can also generate reports using the Dictionary.Indexes table

```
proc sql;
   'select*'
        from dictionary.indexes
          where libname='SASUSER' and
memname='SALE2000';
   quit;
```

Points to Remember

- If you use direct-access read mode to create a representative sample of a data set, you must use a STOP statement to prevent the DATA step from looping continuously.
- An index can enable SAS to more efficiently access specific observations of a data set. However, because indexes use system resources, they should be created only on variables that are commonly used in a WHERE condition or on variables that are used to combine SAS data sets.
- An index is associated with a data set but is stored as a separate file. You can use PROC DATASETS or the CONTENTS statement to generate a report on a data set's indexes. You can also right-click on a data set in SAS Explorer and select **view columns** to view a list of the data set's indexes. You should view this information after you have performed maintenance tasks on your data set to ensure that the index file has been maintained.

Quiz

Select the best answer for each question. After completing the quiz, check your answers using the answer key in the appendix.

1. The variable that is created by the POINT= option is assigned a value ?

a. automatically during compilation of the DATA step.

b. automatically during execution of the DATA step.

c. during compilation of the DATA step, by program statements.

d. during execution of the DATA step, by program statements.

2. Which of the following programs correctly creates a systematic sample from a data set with an unknown number of observations and outputs these sample observations to a data set named *Sample?* ?

a.
```
a.  data sample;
      set sasuser.sale2000 point=thisone nobs=totnum;
      output;
      stop;
   run;
```

b.
```
b. data sample;
     do thisone=100 to totnum by 100;
        set sasuser.sale2000 point=thisone nobs=totnum;
        output;
     end;
     stop;
  run;
```

c.
```
c.  data sample;
     do thisone=100 to 1000 by 100;
        set sasuser.sale2000 point=thisone;
        output;
     end;
     stop;
  run;
```

d.
```
d.  data sample;
     do thisone=100 to totnum by 100;
        set sasuser.sale2000 point=thisone nobs=totnum;
     end;
   run;
```

3. Which of the following expressions will generate a random integer between 1 and 50? ?

a. `ceil(ranuni(50))`

b. `ranuni(50)`

c. `ceil(ranuni(0)*50)`

d. `ceil(ranuni(0))*50`

4. An index ?

a. is an optional file that is associated with a data set.

b. provides direct access to specific observations of a data set, based on the value of one or more key variables.

c. can be classified as simple or composite, either of which can consist of unique values.

d. all of the above

5. Which of the following correctly creates a data set named *Flights* from the *Sasuser.Revenue* data set, creates a composite index named *Fromto* that is based on the values of **`Origin`** and **`Dest`**, and prints informational messages about the index to the SAS log? ?

a.
```
a.  options msglevel=i;
    data flights index=(Fromto=origin dest);
      set sasuser.revenue;
   run
```

b.
```
b.  options msglevel=n;
    data flights (index=(Fromto=origin dest));
      set sasuser.revenue;
  run;
```

c.
```
c. options msglevel=i;
   data flights (index=(Fromto=(origin dest)));
     set sasuser.revenue;
 run;
```

d.
```
d. options msglevel=n;
   data flights (index=Fromto);
     set sasuser.revenue;
  run;
```

6. Which of the following is true? ?

a. When you add observations to a data set, the index(es) are automatically updated with additional value/identifier pairs.

b. When you rename a variable that is used as the key variable in a simple index, you must re-create the index.

c. When you delete a data set, the index file remains until you delete it as well.

d. When you copy a data set with the COPY statement, you must also copy the index file in another step.

7. To create an index on an existing data set, you use ?

a. PROCDATASETS.

b. PROC SQL.

c. the DATA step with the INDEX= option, to rebuild the data set.

d. any of the above

8. Which of the following correctly creates a simple index named *Origin* on the *Revenue* data set? ?

a.
```
a.  proc sql;
      create index origin on revenue(origin);
    quit;
```

b.
```
b.  proc sql;
      modify revenue;
      index=origin;
    quit;
```

c.
```
c.  proc sql data=revenue;
      create index origin;
    quit;
```

d.
```
d.  proc sql;
      index=origin on revenue;
    quit;
```

9. To view a list of the indexes that are associated with a data set, you use ?

a. PROC COPY or the COPY statement in PROC DATASETS.

b. PROC CONTENTS or the CONTENTS statement in PROC DATASETS.

c. the MSGLEVEL= system option and a PROC PRINT step.

d. any of the above

10. Suppose that the *Sasuser.Revenue* data set has a simple index named *FlightID*. For which of the following programs will the index be used? ?

a.
```
a. proc print data=sasuser.revenue;
     where flightid ne 'IA11200';
   run;
```

b.
```
b.  data someflights;
      set sasuser.revenue;
      where flightid > 'IA11200';
   run;
```

c.
```
c.  data someflights;
      set sasuser.revenue;
      if flightid > 'IA11200';
   run;
```

d.
```
d.  proc print data=sasuser.revenue;
      where origin='RDU' or flightid='IA03400';
   run;
```

Answers

1. Correct answer: d

The POINT= option in the SET statement names a variable. You must use program statements to assign a value to this variable during execution of the DATA step, before execution of the SET statement. Also, the value of the POINT=

SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

variable should be a number that corresponds to an observation number in the input data set, and it should be different each time the SET statement executes.

2. Correct answer: b

   To create a systematic sample from a data set that has an unknown number of observations, you use the NOBS=option in conjunction with the POINT= option in the SET statement. The NOBS=variable is automatically assigned a value of the total number of observations in the input data set, and you must assign a value to the POINT= variable before the SET statement executes.

3. Correct answer: c

   In order to create a random sample of a data set, you need to generate a random integer. You can use the RANUNI function in conjunction with the CEIL function to create a random integer. You can use a multiplier with the RANUNI function to increase the range from which the random number is chosen to include as many numbers as you need.

4. Correct answer: d

   An index is a separate file from a data set that contains information about observations within the data set. Specifically, an index contains value/identifier pairs that indicate the location of observations within the data set and the value of one or more key variables in that observation.

5. Correct answer: c

   To create an index at the same time that you create a data set, you use the INDEX= option in the DATA statement. You must assign a unique name to a composite index, while a simple index is automatically assigned the name of the key variable as its name. You can set the value of the MSGLEVEL= system option to I in order to see messages about indexes in the SAS log.

6. Correct answer: a

   For many maintenance tasks that you perform on a data set, SAS automatically performs corresponding tasks to the index file. For example, if you delete a data set, the index file is deleted as well. If you rename a data set with the CHANGE statement in the DATASETS procedure, SAS automatically renames the index file. If you copy a data set to a new location with the COPY statement in the DATASETS procedure, SAS automatically reconstructs the index file in the new location.

7. Correct answer: d

   You can use the DATASETS procedure or the SQL procedure to create an index on or delete an index from an existing data set. You can also rebuild the index with a DATA step and use the INDEX= option to create an index on the rebuilt data set. However, rebuilding a data set uses more system resources than adding an index to an existing data set with either the DATASETS or the SQL procedure.

8. Correct answer: a

   You use the CREATE INDEX statement of the SQL procedure to create an index on an existing data set. In the SQL procedure, you must name the index in the CREATE INDEX statement; for a simple index, the index name must match the name of the key variable.

9. Correct answer: b

   You can use either the CONTENTS procedure or the CONTENTS statement in the DATASETS procedure to generate a list of information about a data set, including a list of existing indexes. All indexes for a data set are stored in a single tile that is separate from but has the same name as the data set.

SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited

10. Correct answer: b

    An index can improve the efficiency with which SAS is able to access certain observations in a data set. However, an index is not always useful. SAS will not use an index to process subsetting IF statements, or other statements that SAS determines might be more efficiently processed without an index.

SAS Institute, SAS Institute Inc. (c) 2011, Copying Prohibited